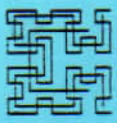
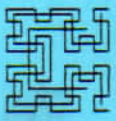


HISOFT DEVPAC



Programmer's Manual

GENS2/ MONS2

The following document details the essential differences between DEVPAC 1 and DEVPAC 2; it should be used in conjunction with the DEVPAC 1 manual with which you should have been supplied.

The complete DEVPAC 2 manual will be available soon at a price of £2 plus VAT.

It is advisable to have the DEVPAC 1 manual handy when using this document since it simply gives the parts of the DEVPAC 1 manual that should be changed or added to. Please read both this booklet and the DEVPAC 1 manual carefully before attempting to use DEVPAC 2, it is a powerful, advanced piece of software.

Please follow these instructions in relation to the DEVPAC 1 manual:

Page 1 – delete and replace with:

GENS2 is a powerful and easy-to-use Z80 assembler which is very close to the standard Zilog assembler in definition. Unlike many other assemblers available for the ZX SPECTRUM, GENS2 is an extensive, professional piece of software and you are urged to study the following sections, together with the example in Appendix 3, very carefully before attempting to use GENS2.

GENS2 is roughly 7K bytes in length and uses its own internal stack so that it is a self-contained piece of software. It contains its own integral line editor which places the textfile immediately after the GENS2 code while the assembler's symbol table is created after the textfile. Thus when loading GENS2 you must allow enough room to include the assembler itself and the maximum symbol table and text size that you are likely you use in the current session. It will often be convenient, therefore, to load GENS2 into low memory.

To load GENS2 proceed as follows:

Place the supplied tape in your cassette recorder, type 'LOAD "GENS2" CODE xxxxx' and press PLAY on the recorder – where 'xxxxx' is the decimal address at which you want GENS2 to run.

Once you have loaded the GENS2 code into the computer you may enter the assembler by 'RANDOMIZE USR xxxxx' where 'xxxxx' is the address at which you loaded the assembler code. If at any subsequent time you wish to re-enter the assembler then"you should execute address 'xxxxx + 50' for a cold start (destroying any text) or address 'xxxxx + 55' for a warm start (preserving any previously created textfile). Note that you should only enter GENS2 once via address 'xxxxx' since this performs the relocation of the master code.

For example, say you want to load GENS2 so that it executes from address £5E00 (or 24064 decimal) – proceed as follows:

```
LOAD "GENS2" CODE 24064
RANDOMIZE USR 24064
```

To re-enter the assembler use RANDOMIZE USR 24114 for a cold start and RANDOMIZE USR 24119 for a warm start.

Once you have entered GENS2 for the first time via address xxxxx, you will first be prompted with the message 'Buffer size? ': you should enter a number between 0 and 9, inclusive, followed by ENTER, or default by simply hitting ENTER alone, in which case the number 4 will be assumed. The number you enter, or default, is taken as the size of the 'Include' buffer in 256 byte units (see Section 2.8 for more details on

the 'Include' option). Thus, if you want to minimise the space occupied by GENS2 and its workspace (and don't care too much about using the 'Include' option efficiently) then you should reply to the prompt with '0' and then ENTER - this will assign the smallest possible 'Include' buffer, which is 64 bytes.

After you have replied to the 'Buffer size? ' question you will be prompted with a '>' sign, the editor's command prompt - consult Section 3 for how to enter and edit text and Section 2 for what to enter.

Important Note: Throughout this manual the '£' sign that denotes a hexadecimal number is actually reached by the hash sign (SYMBOL SHIFT 3) on the SPECTRUM. DO NOT use the '£' sign (SYMBOL SHIFT X).

Page 3 - add the following after line 10.

Note that when using the 'Include' option you may have to specify a larger than normal symbol table size; the assembler cannot predict the size of the file that will be included.

Page 4 - delete lines 21-22 inclusive and insert:

The *C assembler command may be used to produce a shorter assembly listing line - its effect is to omit the 9 characters representing the object code of the line thus enabling most assembler lines to fit on one screen line. See Section 2.8 below.

It is possible to modify the form in which each line of the listing is split by POKEing 3 locations within GENS2. Details of how to do this are given below. We distinguish between 'assembly line' which is the current line of the assembly listing held in an internal buffer and 'screen line' which is a line that actually appears on the SPECTRUM screen. An assembly line will normally generate more than one screen line.

1. Location 'Start of GENS + 3995 (£F9B)' dictates at which column position - 5 the first screen line of the assembly line will be terminated. Change this byte to zero to cause the line to wrap round (useful if you have a full-width printer) or any other value (<256) to end the first screen line at a particular column.

2. Location 'Start of GENS + 4012 (£FB8)' gives the column position (starting at 1) at which each subsequent screen line of the assembly line is to start.

3. Location 'Start of GENS + 4017 (£FBD)' gives how many characters from the remainder of the assembly line are to be displayed on each screen line after the first screen line.

As an example, say you wanted the first screen line of each assembly line to contain 20 characters (i.e. not including the label field) and then each subsequent screen line to start at column 1 and fill the whole line. Also assume that you have loaded GENS at £5E00 or 24064 decimal. To effect these changes, execute the following POKE instructions from within BASIC:

```
POKE 28059,20
POKE 28088,1        there must be at least one space at the
POKE 28093,31       start of each subsequent screen line.
```

The above modifications are only applicable if the *C command has not been used - use of the *C command causes lines to roll over where necessary.

The assembly listing may be paused at the end of a line by hitting CAPS SHIFT and SPACE together - subsequently hit 'E' to return to the editor or any other key to continue the listing.

Page 5 - add the following after line 24.

Spaces and tab characters are treated identically.

Labels may be present alone in an assembler statement; useful for increasing the readability of the listing.

Page 11 - add the following after line 14.

*C-

Shorten the assembler listing starting from the next line. The listing is abbreviated by not including the display of the object code generated by the current line - this saves 9 characters and enables most assembler lines to fit within one 32 character screen line, thus improving readability.

*C+

Revert to the full assembler listing as described in Section 2.0.

*F {filename}

This is a very powerful command which allows you to assemble text from tape - the textfile is read from the tape into a buffer, a block at a time, and then assembled from the buffer; this allows you to create large amounts of object code since the text being assembled does not take up valuable memory space.

The filename (up to 10 characters) of the textfile you wish to 'include' at this point in the assembly may, optionally, be specified after the 'F' and must be preceded with a space. If no filename is given then the first textfile found on the tape is included.

Any textfile that you wish to include via this option must have been previously dumped to tape using the editor's 'T' command and not the 'P' command - this is necessary because a textfile to be included must be dumped out in blocks with sufficient length inter-block gaps to allow the assembly of the current block before the next block is loaded from the tape. The size of the block used by this command (and the editor's 'T' command) is determined on the initial entry to GENS2 from your response to the 'Buffer size? ' prompt, see Section 1. The number you enter (from 0 to 9) following this question is taken as the size of the include buffer in 256 byte units with a default of 4*256 bytes. The ability to select the size of this buffer enables you to optimise the size/speed ratio of any inclusion of text from tape; for example, if you are not intending to use the 'F' command during an

assembly then you may find it useful to specify a buffer size of 0 to minimise the space taken up by GENS2 and its workspace.

Note that the buffer size specified in the session in which you dumped out a file to be included must be the same as the buffer size given in the session in which you are actually including the text.

Whenever the assembler detects an 'F' command it asks you to 'Start tape..', this will happen in the first and second passes since the include text must be scanned in each pass. The tape is then searched for an include file with the required filename, or for the first file. If an include file is found whose filename does not match that required then the message 'Found filename' is displayed and searching continues, otherwise 'Using filename' is displayed, the file loaded, block by block, and included.

See Appendix 3 for an example of the use of this command.

Assembler commands, other than *F, are recognised only within the second pass.

If assembly has been turned off by one of the conditional pseudo-mnemonics then the effect of any assembler command is also turned off.

Page 13 - insert the following after line 5.

In order to reduce the size of the textfile, a certain amount of compression of spaces is performed by the editor. This takes place according to the following scheme: whenever a line is typed in from the keyboard it is entered, character by character into a buffer internal to the assembler; then, when the line is finished (i.e. you hit RETURN), it is transferred from the buffer into the textfile. It is during this transfer that certain spaces are compressed: the line is scanned from its first character, if this is a space then a tab character is entered into the textfile and all subsequent spaces are skipped. If the first character is not a space then characters are transferred from the buffer to the textfile until a space is detected whereupon the action taken is the same as if the next character was the first character in the line. This is then repeated a further time with the result that tab characters are inserted at the front of the line or between the label and the mnemonic and between the mnemonic and the operands and between the operands and any comment. Of course, if any carriage return (RETURN) character is detected at any time then the transfer is finished and control returned to the editor.

This compression process is transparent to the user who may simply use tab control characters (CI - see below) to produce a neatly tabulated textfile which, at the same time, is economic on storage.

Note that spaces are not compressed within comments and spaces should not be present within a label, mnemonic or operand field.

Page 14 - insert the following after line 13:

If, during text insertion, the editor detects that the end of text is nearing the top of RAM it displays the message 'Bad Memory!'. This indicates that no more text can be inserted and that the current textfile, or at least part of it, should be saved to tape for later retrieval.

Page 17 – delete Section 3.2.4 and insert:

3.2.4 Tape Commands.

Text may be saved to or loaded from tape using the commands 'P', 'T' and 'G'.

Command: P n,m,s

The line range defined by n < x < m is saved to tape under the filename specified by the string s. Remember that these arguments may have been set by a previous command. Before entering this command make sure that your tape recorder is switched on and in RECORD mode. Do not use this command if you wish, at a later stage, to 'Include' the text – use the 'T' command instead.

Command: G,,s

The tape is searched for a file with a filename of s; when found, it is loaded at the end of the current text. If a null string is specified as the filename then the first textfile on the tape is loaded.

After you have entered the 'G' command, the message 'Start tape..' is displayed – you should now press PLAY on your recorder. A search is now made for a textfile with the specified filename, or the first textfile if a null filename is given. If a match is made then the message 'Using filename' is displayed, otherwise 'Found filename' is shown and the search of the tape continues.

Note that if any textfile is already present in the memory then the textfile that is loaded from tape will be appended to the existing file and the whole file will be renumbered starting with line 1 in steps of 1.

Command: Tn,m,s

Dump out a block of text, between the line numbers n and m inclusive, to tape in a format suitable for inclusion at a later stage via the assembler command *F – see Section 2.8. The file is dumped with the filename s. The dump takes place immediately you have pressed RETURN so you should ensure that your tape recorder is ready to record before entering this command line.

Note that you should not use this command if you simply wish to append the text later, neither should you use the 'P' command if you wish to 'include' the text.

Page 18 – delete lines 12–15 inclusive and insert:

Command: C

This command allows you to convert textfiles produced by GENS1 to the compressed text format of GENS2. Simply load up the GENS1 textfile, using the 'G' command, use the 'C' command to convert the file and then dump the compressed file out using the 'P' command.

'C' takes no arguments and may take a substantial time to complete the conversion of the file.

Command: V

The 'V' command displays the current values of N1, N2, S1 and S2 i.e. the two default line numbers and the default strings. This is useful before entering any command in which you are going to use default values, to check that these values are correct.

Command: W n,m

The 'W' command causes the section of text between lines n and m inclusive to be output to the printer. If both n and m are defaulted then the whole textfile will be printed. The printing will pause after the number of lines set by the 'K' command - press any key to continue printing.

Command: X

'X' simply causes the start and end address of the textfile to be displayed in decimal. This is useful if you wish to save the text from within BASIC, or if you want to see how much memory you have left after the textfile. GENS2 always expects the text to start at the first address given by the 'X' command and holds the end address of the text in location TEXTEND which is at 'Start of GENS2 + 6434'. Thus, if you wish to 'patch in' a textfile (perhaps produced by MONS2) you must move the textfile to the address specified by the first address displayed by the 'X' command, modify TEXTEND to contain the end address of the file and finally enter GENS2 via a warm start. For example, say you have generated a textfile in the correct place and that it ends (the address after the final end-of-line marker) at £9A02. Then, assuming that you have loaded GENS2 at 24064, you should, from BASIC, POKE 24064+6434,2 (£02) and POKE 24064+6435,154 (£9A) and then enter GENS2 by RANDOMIZE USR 24119. You will now be able to work with the textfile normally from within the editor.

Page 21 - add, at the bottom of the page:

Bad Memory!	This is displayed if there is no room for any more text to be inserted i.e. the end of text is near the top of RAM. You should save the current textfile, or part of it, to tape.

Page 22 - delete the last line and add:

*D *E *H *L *S *C *F

After page 22 add the following new Appendix.

APPENDIX 3 A WORKED EXAMPLE.

There follows an example of a typical session using GENS2 - if you are a newcomer to the world of assembler programs or if you are simply a little unsure how to use the editor/assembler then we urge you to work through this example carefully.

Session objective:

To write and test a fast integer multiply routine, the text of which is to be saved to tape using the editor's 'T' command so that it can easily be 'included' in future programs.

Session workplan:

1. Write the multiply routine as a subroutine and save it to tape using the editor's 'P' command so that it can be easily retrieved and edited during this session, should bugs be present.

2. De-bug the multiply subroutine, editing as necessary.

3. Save the de-bugged routine to tape, using the editor's 'T' command so that the routine may be 'included' in other programs.

Stage 1 - write the integer multiply routine.

We use the editor's 'I' command to insert the text using CI (the tab character) to obtain a tabulated listing. We do not need to use CI, a list of the text will always perform the tabulation for us. We have not indicated where CI's have been used below but you can assume that they are used before the mnemonic and between the mnemonic and the operand.

```
>I10,10 RETURN
   10 ;A fast integer multiply RETURN
   20 ;routine. Multiplies HL RETURN
   30 ;by DE. Returns the result RETURN
   40 ;in HL. C flag set on an RETURN
   50 ;overflow. RETURN
   60 RETURN
   70          ORG   £7E00 RETURN
   80 RETURN
   90 Mult     OR    A RETURN
  100          SBC   HL,DE ;HL>DE?RETURN
  110          ADD   HL,DE RETURN
  120          JR    NC,Mu1 ;yes RETURN
  130          EX    DE,HL RETURN
  140 Mu1      OR    D RETURN
  150          SCF ;overflow ifRETURN
  160          RET   NZ ;DE>255 RETURN
  170          OR    E ;times 0?RETURN
  180          LD    E,D RETURN
  190          JR    NZ,MU4 ;no RETURN
  200          EX    DE,HL ;0 RETURN
  210          RET RETURN
  220 RETURN
  230 ;Main routine. RETURN
  240 RETURN
  250 Mu2      EX    DE,HL RETURN
  260          ADD   HL,DE RETURN
  270          EX    DE,HL RETURN
  280 Mu3      ADD   HL,HL RETURN
  290          RET   C ;overflow RETURN
  300 Mu4      RRA RETURN
  310          JR    NC,Mu3 RETURN
  320          OR    A RETURN
  330          JR    NZ,Mu2 RETURN
  340          ADD   HL,DE RETURN
  350          RET RETURN
  360 CC
>P10,350,Mult
>
```

The above will create the text of the routine and save it to tape. Remember to have your tape recorder running and in RECORD mode before issuing the 'P' command.

Stage 2 - de-bug the routine.

First, let's see if the text assembles correctly. We will use option 6 so that no listing is produced and no object code generated.

```
>A RETURN
Table size: RETURN      {default the symbol table size}
Options: 6 RETURN

*HISOFT GENS2 ASSEMBLER*
 Copyright Hisoft 1983
  All Rights Reserved

Pass 1 errors: 00

Pass 2 errors: 00

*WARNING* MU4   absent
Table used:     74  from    162
>
```

We see from this assembly that we have made a mistake in line 190 and entered MU4 instead of Mu4 which is the label we wish to branch to. So edit line 190:

```
>F190,190,MU4,Mu4 RETURN
  190         JR    NZ,   {now use the 'S' sub-command}
>
```

Now assemble the text again and you should find that it assembles without errors. So now we must write some code to test the routine:

```
>N300,10 RETURN          {renumber so that we can write some more text}
>I10,10 RETURN
   10 ;Some code to test RETURN
   20 ;the Mult routine. RETURN
   30 RETURN
   40          LD    HL,50 RETURN
   50          LD    DE,20 RETURN
   60          CALL  Mult ;Multiply RETURN
   70          LD    A,H ;o/p result RETURN
   80          CALL  Aout
   90          LD    A,L
  100          CALL  Aout RETURN
  110          RET ;Return to editor RETURN
  120 RETURN
  130 ;Routine to o/p A in hex RETURN
  140 RETURN
  150 Aout     PUSH AF RETURN
  160          RRCA RETURN
  170          RRCA RETURN
  180          RRCA RETURN
  190          RRCA RETURN
  200          CALL Nibble RETURN
  210          POP  AF RETURN
  220 Nibble   AND  %1111 RETURN
  230          ADD  A,£90 RETURN
  240          DAA RETURN
  250          ADC  A,£40 RETURN
  260          DAA RETURN
  270          LD    IY,£5C3A ;for ROM RETURN
  280          RST   £10 ;ROM call RETURN
  290          RET RETURN
  300 CC
>
```

Now assemble the test routine and the Mult routine together.

```
>A
Table size: RETURN
Options: 6 RETURN

*HISOFT GENS2 ASSEMBLER*
 Copyright HISOFT 1983
  All rights reserved

709D  190         RECA
*ERROR* 02      (hit any key to continue)

Pass 1 errors: 01

Table used:     88  from    208
>
```

We have an error in our routine; RECA should be RRCA in line 190. So:

```
>E190
  190         RECA
  190 _________C(enter change mode)R RETURN RETURN
>
```

Now assemble again, using simply option 4 (no list), and the text should assemble correctly. Assuming it does, we are now in a position to test the working of our Mult routine so we need to tell the editor where it can execute the code from. We do this with the ENT directive:

```
>35         ENT   $ RETURN
```

Now assemble the text again and the assembly should terminate correctly with the messages:

```
Table used:     88  from    209
Executes: 41105
>
```

or something similar. Now we can run our code using the editor's 'R' command. We should expect it to multiply 50 by 20 producing 1000 which is £3E8 in hexadecimal.

```
>R
0032>
```

It doesn't work! Why? List the lines 380 to 500 (L380,500). You will see that at line 430 the instruction is an OR D followed, effectively, by a RET NZ. What this is doing is a logical OR between the D register and the accumulator A and returning with an error flag set (the C flag) if the result is non-zero. The object of this is to ensure that DE<256 so that the multiplication does not overflow - it does this by checking that D is zero ... but the OR will only work correctly in this case if the accumulator A is zero to start with, and we have no guarantee that this is so. We must ensure that A is zero before doing the OR D, otherwise we will get unpredictable overflow with the higher number returned as the result. From inspection of the code we see that the OR A at line 380 could be made into a XOR A thus setting the flags for the SBC HL,DE instruction and setting A to zero. So:

```
>E380
  380 Mult    OR    A
  380 _______I(enter insert mode)X RETURN RETURN
>
```

Now assemble again (option 4) and run the code, using 'R'. The answer should now be

correct - £3E8.
We can further check the routine by editing lines 40 and 50 to multiply different numbers and then assembling and running - you should find that the routine works perfectly.

Now we have perfected the routine we can save it to tape in 'Include' format:

```
>T300,999,Mult RETURN
```

Remember to start the recorder in RECORD mode before pressing RETURN. Once the routine has been saved like this it may be included in a program as shown below:

```
500         RET
510
520 ;Include the Mult routine here.
530
540 *F Mult
550
560 ;The next routine.
```

When the above text is assembled the assembler will ask you to 'Start tape..' when it gets to line 540 on both the first and second pass. Therefore you should have the Mult dump cued up on the tape in both cases. This will normally mean rewinding the tape after the first pass. You could record two dumps of Mult on the tape, following each other, and use one for the first pass and the other for the second pass.
Please study the above example carefully and try it out for yourself.

Page 25 - delete lines 12-25 inclusive, and insert:

MONS2 is well under 5K in length once it has been relocated but you should allow 5086 bytes on loading MONS2 owing to the table of relocation addresses which comes after the main code. MONS2 contains its own internal stack so that it is a self-contained program.

Once you have entered MONS2 the message '*MONS2 © Copyright Hisoft 1983*' will appear for a few seconds to be replaced by a 'front panel' display (see the Appendix for an example display). This consists of the Z80 registers and flags together with their contents plus a 24 byte section of memory centred (using '>' and '<') around the current value of the Memory Pointer which is initially set to £6000 (24576 decimal). On the top line of the display is a dis-assembly of the instruction addressed by the Memory Pointer.

On entry to MONS2, all the addresses displayed within the Front Panel are given in hexadecimal format (i.e. to base 16); you can change this so that the addresses are shown in decimal by using the command SYMBOL SHIFT 3 - see the next section. Note, however, that addresses must always be entered in hexadecimal.

Commands are entered from the keyboard in response to the prompt '>' under the memory display and may be entered in upper or lower case. Some commands, whose effect might be disastrous if used in error, require you to press SYMBOL SHIFT as well as the command letter. Throughout this manual the use of the SYMBOL SHIFT key may be represented by the symbol '^' e.g. ^Z means hold the SYMBOL SHIFT and Z key down together.

Page 27 - insert after line 4:

IMPORTANT NOTE.

Throughout the rest of this manual, it should be noted that the commands 'K', 'R', 'S' and 'Z' are now reached by holding SYMBOL SHIFT down as well as the relevant command

letter e.g. the 'continue' command is activated by SYMBOL SHIFT and 'K' etc.

SYMBOL SHIFT 3

flip the number base in which addresses are displayed between base 16 (hexadecimal) and base 10 (denary). On entry to MONS2, addresses are shown in hexadecimal, use ^3 to flip to a decimal display and ^3 again to revert to the hexadecimal format. This affects all addresses displayed by MONS2 including those generated by the dis-assembler but it does not change the display of memory contents - this is always given in hexadecimal.

Page 28 - delete line 4-10 and insert:

'H'

convert a decimal number to its hexadecimal equivalent.

You are prompted with ':' to enter a decimal number terminated by any non-digit (i.e. any character other than 0..9 inclusive). Once the number has been terminated, an '=' sign is displayed on the same line followed by the hexadecimal equivalent of the decimal number. Now hit any key to return to the command mode. Example:

H:41472_=A200 here a space was used as the terminator.

Page 28 - insert after line 26:

You may abort this command before you terminate the address by using CAPS SHIFT 5.

Page 32 - delete from line 10 to the bottom of the page, and insert:

each label generated. If you default by simply hitting ENTER then an address of £6000 (hex) is assumed.

Now you are prompted with 'Text:' to enter, in hexadecimal, the start address of any textfile that you wish the dis-assembler to produce. If you do not want a textfile to be generated then simply press ENTER after this prompt. If you specify an address then a textfile of the dis-assembly will be produced, starting at that address, in a form suitable for use by GENS2. If you want to use a textfile with GENS2 then you must either generate it at, or move it to, the first address given by the assembler editor's 'X' command because this is the address of the start of the text expected by GENS2. You must also tell GENS2 where the end of the textfile is; do this by taking the 'End of text' address given by the dis-assembler (see below) and patching it into the TEXTEND location of GENS2 - see the GENS2 manual, Section 3.2. Then you must enter GENS2 by the warm start entry point, to preserve the text.

If, at any stage when you are generating a textfile, the text would overwrite MONS2 then the dis-assembly is aborted - press any key to return to the Front Panel.

After 'Text:', you are asked repeatedly for the 'First:' and 'Last:' (inclusive) addresses of any data areas that exist within the block that you wish to dis-assemble. Data areas are areas of, say, text that you do not wish to be interpreted as Z80 instructions - instead these data areas cause DEFB assembler directives to be generated by the dis-assembler. If the value of the data byte is between 32 and 127 (£20 and £7F) inclusive then the ASCII interpretaion of the byte is given e.g. £41 is changed to "A" after a DEFB. When you have finished specifying data areas, or if you do not wish to specify any, simply type ENTER in response to both prompts.

The screen will now be cleared and there will be a short delay (depending on how

large a section of memory you wish to dis-assemble) while the symbol table is constructed. This having been done, the dis-assembly listing will appear on the screen or printer - you may pause the listing at the end of a line by hitting any key, subsequently hit 'M' (capital 'M' only) to return to the 'front panel' display or any other key (except CAPS SHIFT 1) to continue the dis-assembly. If an invalid opcode is encountered then it is dis-assembled as NOP and flagged with an asterisk '*' after the opcode in the listing.

At the end of the dis-assembly the display will pause and, if you have asked for a textfile to be produced, the message 'End of text xxxxx' will be displayed; xxxxx is the address (in hexadecimal or decimal) that should be POKEd (low order byte first) into the GENS2 location TEXTEND in order that the assembler can pick up this dis-assembled textfile on a warm start. When the dis-assembly has finished, press any key to return to the 'front panel' display, apart from CAPS SHIFT 1 which will return you to BASIC.

Labels are generated, where relevant (e.g. in C30078), in the form LXXXX where 'XXXX' is the absolute hex address of the label, but only if the address concerned is within the limits of the dis-assembly. If the address lies outside this range then a label is not generated, simply the hexadecimal or decimal address is given. For example, if we were dis-assembling between £7000 and £8000, then the instruction C30078 would be dis-assembled as JP L7800; on the other hand, if we were dis-assembling between £9000 and £9800 then the C30078 instruction would be dis-assembled as JP £7800 or JP 30720 if a decimal display is being used. If a particular address has been referenced in an instruction within the dis-assembly then its label will appear in the label field (before the mnemonic) of the dis-assembly of the instruction at that address <u>but only</u> if the listing is directed to a textfile. Example:

```
T
First:8B ENTER
Last:9E ENTER
Printer?Y
Workspace:9000 ENTER
Text: ENTER
First:95 ENTER
Last:9E ENTER
First: ENTER
Last: ENTER

008B FE16       CP   £16
008D 3801       JR   C,L0090
008F 23         INC  HL
0090 37         SCF
0091 225D5C     LD   (£5C5D),HL
0094 C9         RET
0095 BF524E     DEFB £BF,"R","N"
0098 C4494E     DEFB £C4,"I","N"
009B 4B4559     DEFB "K","E","Y"
009E A4         DEFB £A4
```

<u>Page 37 - delete line 46 and insert:</u>

<u>"" SYMBOL SHIFT P</u>

<u>Page 38 - delete line 1 and insert:</u>

<u>Modifying Memory.</u>

END OF MODIFICATIONS